

# Survey and Taxonomy of Self-Aware and Self-Adaptive Autoscaling Systems in the Cloud

Tao Chen, Rami Bahsoon

Centre of Excellence for Research in Computational Intelligence and Applications

School of Computer Science, University of Birmingham

Edgbaston, Birmingham, UK, B15 2TT

{t.chen,r.bahsoon}@cs.bham.ac.uk

**Abstract**—Autoscaling system can reconfigure cloud-based applications and services, through various cloud software configurations and hardware provisioning, to adapt to the changing environment at runtime. Such a behaviour offers the foundation to achieve elasticity in modern cloud computing paradigm. Given the importance of autoscaling in cloud, computational intelligence has been widely applied for engineering autoscaling system, leading to self-aware, self-adaptive and more dependable runtime scaling. In this paper, we present the brief background and history for autoscaling in the cloud, as well as their associations with self-awareness and self-adaptivity of a system. Subsequently, we conduct detailed survey and taxonomy of the key related work and identify the gaps in this area of research.

**Keywords**—performance modeling, self-aware systems; self-adaptive systems; auto-scaling; cloud computing

## I. INTRODUCTION

From the literal meaning of the word "autoscaling", it is obvious that the process is dynamic and requires the system to adapt subject to the uncertain, changing state of the services being managed and the environment. In such a way, the cloud-based services can be 'expanded' and 'shrink' according to the environmental conditions at runtime. Given that it is almost impossible to access the low level details of cloud-based services (e.g., their codes and algorithms) at runtime, an autoscaling system often consist of two physical parts: a managing part containing the autoscaling logic and a manageable part including services and VMs running in the cloud. The two physical parts are seamlessly and transparently connected for realising the entire autoscaling process. This characteristic has made autoscaling systems well-suited to the broad category of self-adaptive systems [1], and the two parts structure of adaptation is known as the external adaptation [2] [3].

While a number of autoscaling approaches have been proposed in the context of cluster, grid and web applications, cloud autoscaling is still in its infancy due to the unique characteristic of cloud, including scaling with cost in mind for better elasticity, considering dynamic changes of both software configuration and hardware resources and the effects of QoS interference etc. In particular, there has been no work that explicitly explore how self-awareness and the related algorithms can be used and what are the benefits for cloud autoscaling in a principled way.

In this paper, we present the brief background and history for autoscaling in the cloud and self-awareness, as well as

self-adaptivity of a system. Subsequently, we conduct detailed survey and taxonomy of the key related work and identify the gaps in this area of research.

## II. BACKGROUND

### A. Self-Adaptivity and Self-Awareness

The broad category of automatic and adaptive systems aim to deal with the dynamics that the system exhibited without human intervention; but this does not necessarily involve uncertainty, i.e., there are changes related to the system but it is easy to know when they would occur and the extent of these changes. Self-adaptivity, being a sub-category, is a particular capability of the system to handle both dynamics and uncertainty. Here, self-adaptive systems refer to the systems that are able to adjust their behaviours according to the perception of the uncertain environment and its own state. According to the adaptive behaviors, self-adaptivity can be regarded as the following four properties, each of which covers a specific set of goals, as explicitly discussed and categorized in many surveys, e.g., [3]:

- **Self-configuring**

The capability of reconfiguring automatically and dynamically in response to changes by installing, updating, integrating, and composing/decomposing software entities.

- **Self-healing**

This is the capability of discovering, diagnosing, and reacting to disruptions. It can also anticipate potential problems, and accordingly take proper actions to prevent a failure. Self-diagnosing refers to diagnosing errors, faults, and failures, while self-repairing focuses on recovery from them.

- **Self-optimizing**

This is also called self-tuning or self-adjusting, is the capability of managing performance and resource allocation in order to satisfy the requirements of different users. End-to-end response time, throughput, utilisation, and workload are examples of important concerns related to this property.

- **Self-protecting**

This is the capability of detecting security breaches and recovering from their effects. It has two aspects, namely defending the system against malicious attacks,

and anticipating problems and taking actions to avoid them or to mitigate their effects.

Self-awareness, on the other hand, is concerned with the system's ability to acquire knowledge about its current state and the environment. Such knowledge permits better reasoning about the system's adaptive behaviours. Consequently, self-awareness is often seen as the lowest level of abstraction of self-adaptivity [3], and thus it can improve the perception and self-adaptivity of a system [4] [5] [6] [7]. Inspired from the psychology domain, Becker et al. [8] have classified self-awareness of a computing system into the following general capabilities (they have used node to represent any conceptual part of a system being managed):

- **Stimulus-aware**

A node is stimulus-aware if it has knowledge of stimuli. The node is not able to distinguish between the sources of stimuli. It is a prerequisite for all other levels of self-awareness.

- **Interaction-aware**

A node is interaction-aware if it has knowledge that stimuli and its own actions form part of interactions with other nodes and the environment. It has knowledge via feedback loops that its actions can provoke, generate or cause specific reactions from the social or physical environment.

- **Time-aware**

A node is time-aware if it has knowledge of historical and/or likely future phenomena. Implementing time-awareness may involve the node possessing an explicit memory, capabilities of time series modelling and/or anticipation.

- **Goal-aware**

A node is goal-aware if it has knowledge of current goals, objectives, preferences and constraints. It is important to note that there is a difference between a goal existing implicitly in the design of a node, and the node having knowledge of that goal in such a way that it can reason about it. The former does not describe goal-awareness; the latter does.

- **Meta-self-aware**

A node is meta-self-aware if it has knowledge of its own capability(ies) of awareness and the degree of complexity with which the capability(ies) are exercised. Such awareness permits a node to reason about the benefits and costs of maintaining a certain capability of awareness (and degree of complexity with which it exercises this level).

The benefits that self-awareness introduces for computing systems, including better solution for runtime dynamics and uncertainty, heterogeneity, and trade-offs on objectives, have rendered it as a neat solution for the challenges of cloud autoscaling.

## B. Autoscaling in Cloud

Depending on the given QoS attributes and the manageable cloud primitives, an autoscaling system can cover self-configuring, self-healing, self-optimising and self-protecting,

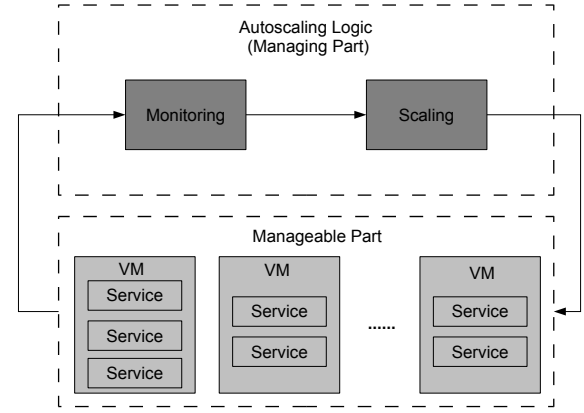


Fig. 1. The Simplest Form of An Autoscaling System.

or any combination of those, from the notion of self-adaptivity. A recent survey [6] has established the evidences that self-awareness can improve a system that requires self-adaptivity, and therefore achieving self-awareness is a promising way to enable better autoscaling systems in cloud.

The external adaptation of an autoscaling system is shown in Figure 1. As we can see, the core of an autoscaling system in the cloud is the autoscaling logic, which can consist of multiple logical aspects. The simplest form of autoscaling system covers monitoring and scaling aspect in its autoscaling logic: the former gathers the service's or application's current state while the latter utilises the information to decide an action. However, such a simplified form of autoscaling system tends to be limited, since it cannot effectively handle the increasing runtime complexity of the cloud environment, including, e.g., dynamic and uncertainty caused by workload, the QoS performance and heterogeneity of cloud-based services. Given the shared infrastructure of cloud, the autoscaling process should be aware of and be able to handle QoS interference. This is because improving the QoS performance of a service may likely to downgrade that of its neighbouring services and VMs, which will negatively affect the overall quality of autoscaling and elasticity.

To improve the quality of adaptation, modern autoscaling systems often additionally cover other more sophisticated aspects, including *modelling*, *determining granularity of control* and *decision making*. The modelling is concerned with the model of QoS, environment conditions (e.g., workload) and demand of the control knobs (e.g., software configurations and hardware resources). The resulting models are a powerful tool to assist the autoscaling decision making process. Without loss of generality, in this paper, we term both control knobs and environment conditions in the cloud as *cloud primitives*. In such context, we further decompose the notion of primitives into two major domains: these are **Control Primitive (CP)** and **Environmental Primitive (EP)**. Control Primitives are the internal control knobs and can be either software or hardware, which can be managed by the cloud providers to support QoS. Specifically, software control primitives are software tactics and the key configurations in cloud; such as the number

of threads in the thread pool of a service/application, the buffer size and load balancing policies etc. Whereas, hardware control primitives are computational resources, such as CPU and memory. Software and hardware control primitives rely on the PaaS and IaaS layers respectively. In particular, it is non-trivial to consider software control primitives when autoscaling in the cloud as they have been shown to be important features for QoS [9] [10] [11]. On the other hand, Environmental Primitives refer to the external stimuli that cause dynamics and uncertainties in the cloud. These, for example, can be the workload and unpredictable incoming data etc. If the cloud provider is able to control the presence of the stimulus, then these can be considered as control primitives. These models can often assist and improve the autoscaling decision making. It is worth noting that the examples of primitives listed above are not exhaustive, Ghanbari et al. [12] have provided a more completed and detailed list of the possible control primitives in cloud.

Determining granularity of control in the autoscaling logic is essential to ensure the benefit (e.g., QoS and cost objectives) for all cloud-based service. It is concerned with understanding whether certain objectives can be considered in isolation with some of the others. This is because *objective-dependency* (i.e., conflicted or harmonic objectives) often exist in the decisions making process, which implies that the overall quality of autoscaling can be significantly affected by the inclusion of conflicted or harmonic objectives in a decision making process, hence rendering it as a complex task. This is especially true for the shared infrastructure of cloud where objective-dependency exists for both intra- and inter-services. That is to say, objective-dependency is not only caused by the nature of objectives (intra- service), e.g., throughput and cost objective of a service; but also by the QoS interference (inter-services) due to the co-located services on a VM and co-hosted VMs on a PM [9] [10] [11] [13].

The final logical aspect in autoscaling logic is the dynamic decision making process that produces the optimal (or near-optimal) decision, which consists of the newly configured values of the related control primitives, for all the related objectives. In the presence of objective dependency, autoscaling decision making requires to resolve complex trade-offs, subject to the SLA and budget requirements. The trade-off decision can be then executed using either vertical scaling and/or horizontal scaling actions, which adapt the cloud-based services and/or VMs correspondingly.

In the following sections, we provide survey and taxonomy for the most influent and recent work that is related to this paper. Particularly, we present the review and discussions based on the key logical aspects for autoscaling in the cloud, which are architectural pattern, QoS modelling, granularity of control and decision making. We then position this paper by discussing how our work differ from those existing approaches.

### III. ARCHITECTURAL PATTERN

Autoscaling architecture is the most essential element of an autoscaling system in the cloud. It describes the structure of the autoscaling process, the interaction between components

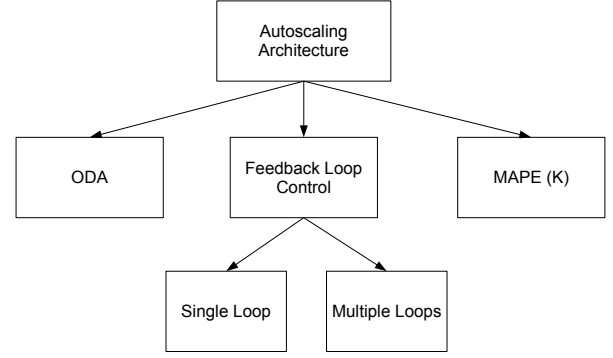


Fig. 2. The Taxonomy of Architectural Patterns used in Autoscaling.

and the modularisation of the important logical aspects in autoscaling. In the following, we survey the key architectural patterns that have been applied for autoscaling in the cloud. In particular, we classify them into three categories based on their basic form; these are Feedback Loop Control [14], Observe-Decide-Act [15] and Monitor-Analysis-Plan-Execute [16], where the latter two are essentially detailed variations of the former one. The classification of those architectural patterns is the result obtained from literature review. As one of the key outputs, we have identified that those three architectural patterns are predominantly applied for autoscaling system in the cloud. The taxonomy has been illustrated in Figure 2.

#### A. Feedback Loop Control

Feedback Loop Control is the most general architectural pattern for controlling self-adaptive systems, including the autoscaling systems. It is usually a closed-form loop made up of the managing system itself and the path transmitting its origin (e.g., a sensor) to its destination (e.g., an actuator). Here, we further divide the pattern in terms of whether single or multiple loops are used.

##### Single Loop Control

Single loop control is the simplest, yet the most commonly used pattern for autoscaling in the cloud due to its flexibility. The most common practice with single loop control is to build a feedback loop where the core is the decision making component and an optional QoS modelling component, e.g., Ferretti et al. [17], CloudOpt [18], SmartSLA [19], Padala et al [20], Kateb et al [21], Jiang et al. [22], CLOUDFARM [23], Grandhi et al. [24]. Some other work has included an additional component for workload or demand prediction based on either offline profiling, e.g., Jiang et al. [25] and Fernandez et al. [26], or online learning, e.g., Kingfisher [27], Gambi et al. [28], Chihi et al. [29] and PRESS [30].

Open feedback loop exists, as presented in Cloudine [31], where the scaling actions are partially triggered by user requests. In particular, they use a centralised *Resource and Execution Manager* to handle all the scaling actions. Apart from the general autoscaling architecture, other efforts are

particularly designed upon specific cloud providers [32], [33], [34], [35]. For example, Zhang et al. [33] and Kabir and Chiu [34] propose to use a simple feedback loop for architecting autoscaling system, which is heavily tied to the properties of Amazon EC2 and S3. Other applications of single loop control, which are worth mentioning, include: VScale [36] is a feedback based framework that particularly focus on vertical scaling of VM in the cloud. It is deployed in a decentralised manner where there is a dedicated instance running on each PM. iBoolean [37] is a feedback control approach for autoscaling hardware resource in the cloud. It is designed as a distributed management framework, in which each individual VM initialise its own management.

There are architectures using single loop control where the core is classical control theory [38], [39], [40], [41]. Particularly, Anglano et al. [38] present a fuzzilized feedback control for autoscaling in the cloud. It is a typical controller using fuzzy theory driven by application performance. Guo et al. [39] also present a fuzzy logic based feedback control. However, it only intend to scale the software control primitives.

### **Multiple Loop Control**

Unlike the single loop control approach, it is possible to use multiple loops and controllers for autoscaling in the cloud. Here, multiple feedback loops operate in different levels of the architecture, e.g., one operates at the cloud level while the others operate on each VM. The benefit is that multiple loops provide low coupling in the design of the loops. Notably, multiple loop control can be used to separate global and local controls [42], [43], [44], [45]. Among others, [42] apply a decentralised feedback control for autoscaling CPU in the cloud. Although it aims for individual applications, the controllers actually operate on each tier of an application. Different controllers do not need to interact with each others. ARUVE [43] utilises a global controller in conjunction with the local controller to form multiple feedback loops. The local controllers are decentralised on each PM while the global controller is centralised.

Multiple loop control is also effective for isolating the logical aspects of autoscaling and management in the cloud [46], [47], [48], [49], [50], [51], [52] [9], [53]. For example, Emeakaroha et al. [50] has also used multiple feedback loops. In particular, they use a global feedback loop consisting of three local feedbacks, each of which operate on SaaS, PaaS and IaaS layer. Wang, Xu and Zhao [51] propose a two layer feedback control for autoscaling in the cloud. The first layer, termed guest-to-host optimisation, controls the hardware resources, e.g., CPU and memory. Subsequently, the host-to-guest optimisation adapts the software configuration accordingly.

Overall, existing work adopts feedback loop control for its simplicity and flexibility. However, instead of designing autoscaling with a clear architectural blueprint beforehand, they utilise a bottom-up approach where the design of autoscaling system starts off from the underlying techniques and algorithms. Such design can limit the consideration of required knowledge for the autoscaling system to perform adaptations,

or the consideration is rather simple and coarse-grained, as they do not express what level of knowledge is required at which logical aspect of the system, and how they can be beneficial for the adaptation.

### **B. Observe-Decide-Act**

Observe-Decide-Act (ODA) loop [15] is considered as an extended pattern of the feedback loop control, and it is concerned with the system monitoring itself and its environment, making decisions about how to adapt behaviour using a set of available actions. A unique *Decide* component separates it from the feedback loop control, as it explicitly requires to perform reasoning about the effects of adaptation on the system's goals and objectives. While an ODA loop is most commonly applied for self-adaptive systems in general, only few work (e.g., [54]) has included it for the design of autoscaling system in the cloud. This is because one important aspect of ODA is to define the effects of human activities on the adaptive behaviours, which is a difficult practice for autoscaling in the cloud.

SEEC [15], being the very first work to introduce ODA, is a general framework for self-aware and self-adaptive systems. It applies ODA for decoupling the loops to different roles (i.e., application developer, system developer, and the SEEC runtime decision infrastructure) in the development life-cycle, each role focuses on one or more steps in ODA. [55] adopt an ODA loop to manage FPGA-based systems, where the decision and the its translation to actions are conducted by an incorporation of the *Decide* and *Act* steps. Bolchini et al. [56] have used ODA to realise the adaptation for self-adaptive systems because of its simplicity. It observes high-level and raw data from the *Observe* step, such data is then used by *Decide* to know which parts of the system to reason on, and finally, actions are taken depends on the characteristic of the system being managed. Huber et al. [54] also use ODA for self-aware autoscaling resources in the cloud. However, unlike traditional ODA loop, it has an additional *Analysis* step which is used to detect the type of problems that trigger adaptation.

Overall, although the knowledge that a system requires is sometime discussed in the *Decide* step (e.g., [54]) in ODA, it cannot capture different levels of knowledge in a fine-grained representation as required by the system. This is because ODA is mainly designed for decoupling loops for different human activities, which allows application and systems programmers to separately specify observations and actions, according to their expertise.

### **C. Monitor-Analyze-Plan-Execute**

Another pattern extended from the feedback loop control, namely Monitor-Analyze-Plan-Execute (MAPE), is firstly proposed by IBM for architecting self-adaptive systems. In such pattern, the *Decide* step in OAD is further divided into two substeps, these are *Analyze* and *Plan*, where the former is particularly designed to determine the causes for adaptations, e.g., SLA violation; the latter, on the other hand, is responsible for reasoning about the possible actions for adaptation. MAPE sometime can be extended by a Knowledge component (a.k.a.

MAPE-K) which maintains historical data and knowledge used by the system for better adaptation.

MAPE (or MAPE-K) is widely applied for autoscaling in the cloud [57], [58], [59], [60], [61], [62]. For example, the architecture of the FoSII project [59] [60] leverages MAPE-K to realise the self- management interface, which is necessary to devise actions in order to prevent SLA violations in cloud. They also use the additional *Knowledge* (K) component to record cases and the related solutions, which can assist the autoscaling decision making. QoS MOS [61] is designed for service-based systems rather than for cloud specific autoscaling, however, it contains many aspects similar to that of autoscaling in the cloud. To achieve continuous adaptation, it applies MAPE with the focuses on analytical QoS modelling and optimisation of resource allocation. APPLEware [62] is an autoscaling framework which leverages MAPE. In their architecture, the *Analyze* component model the QoS while the *Plan* component conducts optimisation process for autoscaling.

Realising multiple MAPE loops is also possible. Zhang et al. [10] introduce an architecture for autoscaling using two nested MAPE loops. The first loop is responsible for adapting the software primitives while the other loop is used to change the hardware primitives. These two loops run sequentially upon autoscaling, that is, adapting the software primitives before changing the hardware primitives. Similarly, BRGA [63] utilises MAPE to realise a framework for autoscaling in the cloud. Such solution consists of both the local and global view of the cloud-based application. In particular, the *Monitor* and *Execution* phase maintain the global view whereas the *Analyze* and *Plan* phase manage the local view on each PM. The authors claim that such an approach can achieve good global quality with reasonable management overhead.

In conclusion, MAPE can be good for separation of concepts (e.g., *Analyze* and *Plan*) and for expressing the sequential interactions between those concepts. However, although the *Knowledge* component can be considered, there is still no fine-grained representation of the required knowledge for the system. Thus, it is not immediately intuitive that what level of the knowledge is required by each logical aspect of the autoscaling system.

#### IV. QoS MODELLING

QoS modelling, or performance modelling, is a fundamental research theme in cloud computing and it can serve as useful foundations for addressing many research problems in the cloud [64], including autoscaling. The QoS models correlate the QoS attributes to various control primitives and environmental primitives. Clearly, these models are particularly important in cloud autoscaling, as they are a powerful tool that can assist the reasoning about the effects of adaptation on objectives in the autoscaling decision making process. Typically, QoS modelling consists of two phases, namely primitives selection and QoS function training. More precisely, the primitives selection phase determines *which* and *when* the primitives correlate with the QoS; while QoS function training phase identify *how* these primitives correlate with the QoS, i.e., their magnitudes in the correlation. The QoS models can be

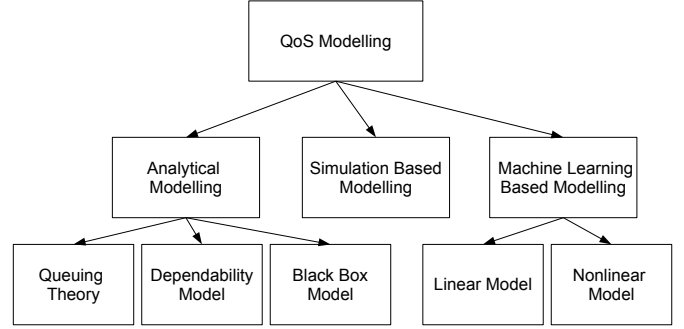


Fig. 3. The Taxonomy of QoS Modelling in Cloud.

either static or dynamic, where the former refers to the models' expression and their structure (e.g., the number of inputs and their weights) do not change over time; while the latter permits such changes. Those models can be also applied as online at system runtime, or offline at design phase of the system. In the following, we survey the key work on QoS modelling in the cloud and classify them in terms of the algorithms they apply. The taxonomy has been illustrated in Figure 3.

##### A. Analytical Modelling

Analytical modelling approaches rely on mathematical models that have a closed-form solution to model the cloud-based service. These models are often built offline based on theoretical principles and assumptions. Next, we further divide the analytical modelling approach into queuing theory, dependability models and black box models.

##### Queuing theory

Queuing model and queuing network are widely applied for QoS modelling in the cloud. They model the cloud-based services as a single queue or a collection of queues interacting through request arrivals and departures. Specifically, a single queue has been used to model the correlation of response time (or throughput) to CPU, number of VM and workload. For example, depending on the assumption of the distribution on arrival and service rate, the model can be built as M/G/c queue by Zhang et al. [33], M/G/m queue by Jiang et al. [22], M/M/1 queue by E<sup>3</sup>-R [65] and JustSAT [66], and M/M/m queue by Jiang et al. [25]. To create more detailed modelling with respect to the internal structure of cloud-based services, multiple queues can be used to create QoS models: Goudarzi and Pedram [67] apply multiple queues to model the response time for cloud-based multi-tiered applications with respect to number of VM and workload. Their work calculates average response time for the queue in the forward direction throughout the tiers. In a similar way, Bi et al. [68] use a queuing network composed of an M/M/c queue and multiple M/M/1 queues to estimate the correlation between response time and number of VMs and workload for cloud-based application. Li et al. [69] apply a single queue to model

the correlation between response time and CPU, workload and thread. In particular, the model contains finite capacity regions, which denote the place constraints on the maximum number of jobs circulating in a subnetwork of queues. This is because they are the simplest class of models that offer the features to describe performance scalability as a function of the software threading level and for the number of CPUs.

Unlike classical queuing model and queuing network, the Layer Queuing Network (LQN) additionally model the dependencies arising in a complex workflow of requests to cloud-based services and applications. Chi et al. [70] use LQN (i.e., based on M/M/n queue) to model the QoS of application, which is response time with respect to CPU and workload. CloudOpt [18] relied on LQN as the aggregate QoS model for all the services contained by an application. It models only response time with respect to CPU and workload. Li et al. [57] use LQN for model services in an application. Again, it only captures response time with respect to CPU and workload. Zhu et al. [35] have also used LQN where the authors employ a global M/M/c queue for the entire on-demand dispatcher and then a M/G/1 queue on each tier of an application. The former queue correlates the response time to number of VMs while the latter queue models the relationship between response time and CPU of the VM that contains the corresponding tier.

### ***Dependability models***

Dependability models are another widely used technique for QoS modelling in the cloud. This approach focuses on the modelling of stable states for QoS attributes. For example, Copil et al. [71] uses a graph representation to model the dependency between per-service QoS and the necessary primitives. Although the graph can be updated at runtime, the model is essentially analytical. In QoSMOS [61], the authors analytically solve the Markov Models (Discrete-Time Markov Chain and Markov Decision Process) to model the QoS for services in an application. The model correlates QoS attributes with hardware resources and workload. Huber et al. [54] uses Palladio Component Model (PCM) as architecture-level QoS model since it allows to explicitly model different usage profiles and resource allocations. Kateb et al. [21] uses *model@runtime* to correlate QoS attributes with the number of VM. The modelling approach is essentially based on a domain specific language, which does not only able to reason about the system at design time, but is also able to assist decision making during runtime.

### ***Black box models***

Black-box models are also popular, in which the QoS is modelled based on empirical knowledge or statistical data of history [50], [59], [23], [72], [73], [74], [34], [63]. Among others, CLOUDFARM [23] uses an empirical QoS model where the correlation between certain QoS values and the required resource is captured (i.e., CPU). In particular, the authors assumed that the magnitudes of resources to the QoS values is known, as specified by the cloud service or application provider. The FoSII project [59] has also applied empirical QoS models such that the correlation between hardware re-

source (i.e., CPU, memory and bandwidth) and QoS is hard-coded using cases, each of which contains a set of particular values of resource and their resulted utility value. Another work from Emeakaroha et al. [72] propose an empirical model that maps the expected QoS values with CPU, memory, bandwidth and storage. The model relies heavily on the assumption of the system that being managed. Their extended work [73] is also based on a similar approach, where the authors correlate the QoS attributes to different CPU, memory, bandwidth and storage using manual and empirical mapping. The proposed mapping can be as simple as one QoS attribute to one primitives, or a complex form where multiple cloud primitives are associated with a QoS attribute.

Overall, analytical modelling has the advantage of simplicity and interoperability. In particular, such modelling is usually highly intuitive and has negligible overhead when applied for autoscaling in the cloud. However, analytical approaches often require in-depth knowledge about the likely behaviours of the system being modelled. Consequently, their effectiveness is restricted to the assumptions of service's internal operations; such static nature makes these approaches limited in coping with the dynamic and uncertainty at runtime. Finally, both primitives selection and QoS function training phases in analytical approaches are often static and offline. However, for some of the approaches (e.g., [61], [21]), their QoS function training phase can be achieved in a dynamic and online manner.

### ***B. Simulation Based Modelling***

Various simulators exist for creating QoS models; here, conducting simulations is usually a complex and expensive process and thus they are used in an offline manner. In practice, simulation is required to be setup by the domain experts, who will often need to analyse, interpret and profile the data collected after simulation runs. Specifically, Fernandez et al [26] have relied on a profiling approach that builds the QoS model for each bundle of VM offline. The process is similar to a simulation modelling approach. CDOSim [75] is a framework that simulates the actual application in the cloud to restrict the search-space for autoscaling and to steer the exploration towards promising decisions. CloudSim [76] is a simulation toolkit that models QoS attributes (of VM) with respect to resource allocation. It supports both single cloud and multiple clouds scenarios. As an extension of CloudSim, CloudAnalysis [77] allows the simulation of QoS attributes for the application deployed on geographically-distributed datacenters. Similarly, DCSim [78] simulates the overall quality of resource autoscaling for the entire cloud.

Overall, simulation can produce good QoS models providing that the scenarios which have been simulated are similar to those that would occur at runtime. However, similar to the analytical approaches, simulation based modelling is also static and restricted by the assumptions made in the simulators, e.g., distribution of workload and the effects of QoS interference. In addition, it can be expensive to use as it often requires heavy human intervention. Commonly, simulation based modelling approach is an offline process, in particular, the QoS function

training phases can be dynamic; while the primitives selection phase is static.

### *C. Machine Learning Based Modelling*

The increasing complexity of managing services in the cloud makes the modelling difficulty far beyond the capability of human analysis. To this end, recent works have been leveraging the advances of machine learning algorithms. In the following, we survey the key work that applies machine learning approaches for QoS modelling in the cloud. In particular, we have classified them into two categories, these are: linear and nonlinear modelling.

#### *Linear modelling*

Learning algorithms based on linear models for QoS modelling in the cloud can handle linear correlation between a selected set of inputs (e.g., CPU, memory, number of VM, workload etc) and output (i.e., QoS attributes), and they are sometime very efficient. Diao et al. [79] propose a very early work on QoS modelling using Auto-Regressive and Moving-Average (ARMA) and Multi-Inputs-Multi-Output (MIMO) model on-the-fly. Their work is not cloud specific but it provides insight for many subsequent work on cloud based QoS modelling. Simple linear models most commonly rely on linear regression, where each primitive input is associated with a time-varying weight, e.g., Lim et al. [40], Zhang et al. [10] and Collazo-Mojica et al. [80]. More advanced forms exist, e.g., Padala et al. [20] have used ARMA trained by Recursive Least Squares (RLS). The authors claim that the linear ARMA model is easy to be estimated online and can simplify the corresponding controller design problem. The authors found that the second-order ARMA model can predict the application performance with adequate accuracy. Kalivianaki et al. [42] uses Kalman filter to update the QoS model. The authors claim that the Kalman filter is optimal in the sum squared error sense under the assumptions that the system is described by a linear model, and the process and measurement noise are white and Gaussian.

Linear machine learning algorithms are also commonly used with analytical approaches to form QoS models. Specifically, Grandhi et al. [24] and Zheng et al. [81] have proposed hybrid model: to model the multi-tiered application, they have relied on a modified LQN where there are some time-varying coefficients. The authors then employ the Kalman filter as an online parameter estimator to continually estimate those coefficients. Ghanbari et al. [82] have also followed a similar approach, but through the use of k-mean clustering, they additionally cluster the model into multiple sub-models based on different types of workload. The approach proposed by Xiong et al. [47] has relied on a combined model, where a M/G/1 queue is used to model the correlation between response time and workload; while ARMA is used to model the relationship of response time and CPU.

There is limited work that attempts to capture the information of QoS interference in the linear QoS model and they only focus on the VM-level [62], [13], [83], [11]. As an example, Q-Cloud [13] has explicitly considered QoS interference by

using the hardware control primitives of all co-hosted VMs as inputs, rendering it in a MIMO manner. The model itself is a simple linear model and it can be easily trained by using Least Mean Square (LMS).

#### *Nonlinear modelling*

Learning algorithms based on nonlinear models for QoS modelling in the cloud is able to capture complex and non-linear correlation, in addition to the linear one. However, it can also produce relatively large overhead than the linear modelling. Here, existing work often aim to model the correlation between hardware control primitives (e.g., CPU, memory and bandwidth) and QoS. The nonlinear modelling can be relied on kriging model [28], Regression Tree (RT) [19], Artificial Neural Network (ANN) [84] [85] [86], Support Vector Machine (SVM) [49] [84], change-point detection [87], and ensemble or bucket of multiple algorithms [88] [89] [90]. For example, Gambi et al. [28] utilise kriging model, which is a spatial data interpolator akin to nonlinear and radial basis functions, and it extends traditional regression with stochastic Gaussian processes. SmartSLA [19] employs Regression Tree (RT) and boosting to model the QoS. RT partitions the parameter space in a top-down fashion, and organises the regions into a tree style. The tree is then trained by M5P where the leaves are regression models. The work from Kunda et al. [84] presents sub-modelling based on ANN and SVM for correlating QoS with hardware control primitives in the cloud. Instead of building a single model for a QoS attribute, they train  $n$  sub-models, whereby  $n$  is determined by performing k-mean clustering based on the similarity between data values of QoS. This is because they observe that large errors were mostly concentrated in a few sub-regions of the output value space, indicating a single model's inability to accurately characterise changes in application behaviour as it moves across critical resource allocation boundaries. The authors claim that the approach can be applied online.

Examples exist for cases where multiple linear and/or non-linear machine learning algorithms are used together. Zhu and Agrawal [53] use a variant of ARMA and SVM to model the correlation of QoS attributes to software and hardware control primitives. In particular, the ARMA variant, which is trained by SVM, is used to link QoS and software control primitives. Subsequently, another dedicated SVM is used to model the relationship between software control primitives and hardware control primitives (i.e., CPU and memory in the work). Another work from Kousiouris et al. [91] correlates QoS attributes with various primitives using ANN. Additionally, it applies a time-series ANN to predict the workload in conjunction with the QoS models. This aims to provide more accurate information when making prediction online.

#### *Dynamic primitives selection*

All the aforementioned work regards the primitives selection as a manual and offline process, most commonly, they have relied on empirical knowledge and heavy human analysis to select the important primitives as the inputs of QoS models. Although not many, there is some work that explicitly considers dynamic process in primitives selection, which tends to be

more accurate and can be easily applied [48], [92], [93]. As an example, vPerfGuard [93] is a framework that correlates QoS attributes with respect to software control primitives, hardware control primitives and environmental primitives. The authors achieve primitive selection based on both filter (relevance based correlation coefficient) and wrapper (i.e., hill-climbing comparison based on algorithms like k-nearest-neighbour and linear regression etc). Linear regression is used as default to train QoS based on the selected primitives.

### Comparison of different learning algorithms

Given the various types of machine learning algorithms, it can be difficult to determine which one(s) are the appropriate algorithms for QoS modelling in the cloud, with respect to both accuracy and overhead. There are researches that have conducted detailed comparisons of different possible learning algorithms for QoS modelling in the cloud [94] [95] [96], for example, Lloyd et al. [95] conduct an extensive experiment over various machine learning algorithms (i.e., MLR, MRS and ANN) in cloud. They select the primitives based on manual analysis. The results show that the relevant and useful primitives could be different depending on the characteristics of services and application; and that different machine learning algorithms achieve a variety of accuracy depending on the scenarios.

Overall, machine learning based modelling approaches have the advantage of requiring limited human intervention, and are able to continually evolve themselves at runtime in order to cope with dynamics and uncertainty. Nevertheless, depending on the learning algorithm, the resulting overhead can be high (e.g., the nonlinear ones) and the accuracy is sensitive to the given scenarios (e.g., fluctuation of the data trend). Generally, the machine learning approaches can be applied as offline, online or a mixture of the both. According to the existing work surveyed for QoS modelling in the cloud, we discover that the QoS function training phase is often dynamic; while there is very little work that intends to consider primitives selection phases as a dynamic process (online or offline), and the others have relied on offline and manual analysis. Therefore, we can conclude that majority of the approaches that apply machine learning for QoS modelling are semi-dynamic. In addition, we have also found that only a small amount of existing work intends to consider QoS interference in the modelling; and they only focus on VM-level interference.

### D. Comparison of QoS Modelling to Workload and Demand Modelling

Despite the fact that QoS modelling is fundamentally helpful for cloud autoscaling, it can be difficult to achieve given the heterogeneity of possible primitives and the multi-dimensional input space of QoS modelling. Therefore, some existing work on autoscaling (e.g., [27], [97], [86]) have considered simpler alternatives, i.e., model the workload and demand for assisting autoscaling decision making. In those cases, the modelling is reduced to a single dimension, where the core is to model the trend of the workload or demand using its historical data.

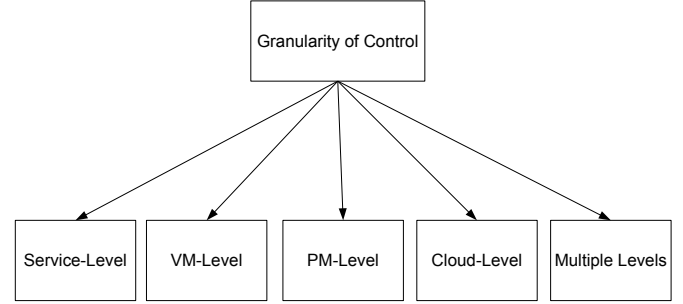


Fig. 4. The Taxonomy of Granularity of Control for Autoscaling in Cloud.

The models would be used to predict the likely value at the next interval. However, the single dimension in workload or demand models do not offer the ability to reason about the effects of autoscaling decisions and the possible trade-offs. It is important to note that trade-off decisions making is a critical logical aspect for autoscaling, and it is also one of the cores of this paper. Therefore, this paper has explicitly focused on QoS modelling.

## V. GRANULARITY OF CONTROL

The ultimate goal of autoscaling is to optimise the QoS and cost objectives, which are referred to as benefit, for all cloud-based services. To this end, the granularity of control in autoscaling plays an integral role, since it determines which and how many objectives should be considered in a decision making process of autoscaling. In the following, we classify existing cloud autoscaling approaches into different categories depending on what level of granularity they tend to operate at. The taxonomy has been illustrated in Figure 4.

### A. Controlling at Service Level

Service level is the finest level of control in the cloud. It is worth noting that by service, we refer to any conceptual part of the system being managed. As a result, control granularity at the service level may refer to independently controlling/scaling an application, a tier of an application or a cloud-based service.

Specifically, most work has focused on controlling each cloud-based application. These approaches have relied on controlling the QoS and/or cost for each individual application in isolation, and therefore, they regard an application as a service. Examples of such include: [27], [66], [26], [80], [53], [69], [22], [58], [30] and [48]. To describe some of them in detail, Lim et al. [40] control the application and its required VM, in which case an application is regarded as a service. Sedaghat et al. [97] regard application as a service, and considered the required number of VMs and the fixed VM bundles for such service. Jiang et al. [25] control each application, each of which is, again, regarded as a service and therefore it is essentially service-level control. In addition, the authors group VMs into different fixed bundles.



There is also existing work that controls cloud-based service in general, which can be regarded as any conceptual part of a cloud-based system. Copli et al. [71] control the QoS, cost and their elasticity for each service deployed in the cloud. Yang et al. [98] control the cost of individual cloud-based services. The FoSII project [59] controls individual cloud-based service, their QoS and cost. [28] control at the service level, where the controller decides on the optimal autoscaling decision for cloud-based service in isolation. QoS MOS [61] explicitly focuses on each individual service, and adapting it in isolation. Kateb et al. [21] consider many services are encapsulated in the application, and the authors focus on each service in isolation. The E<sup>3</sup>-R framework [65] and Frey et al. [99] control each service, including their composition and autoscaling.

#### *B. Controlling at Virtual Machine Level*

VM level means that the control and decision making operate at each VM. In particular, certain work assumes a one-to-one mapping between application (or a tier) and VM and thus they can be categorised as either service level or VM level granularity. To better separate them from the pure service level granularity of control, these work are regarded as VM level granularity. Specifically, FC2Q [38] regards application tier and VM interchangeably, therefore controlling each tier of an application is equivalent to control each individual VM. Similarly, Kalyvianaki et al. [42] control a tier of an application that resides on a VM, and the authors only focus on CPU allocation of a VM. Wang, Xu and Zhao [51] control the cloud in a per-VM basis, and each VM is adapted in isolation. VScale [36] focuses on vertical scaling only and hence the control granularity is per VM. Zhang et al. [10] assume only one application per VM, and control the deployed VM in isolation correspondingly. Guo et al. [39] control the application deployed on the VM and the authors assume one application per VM. Similarly, Matrix [49] has used VM level control as there is only one application per VM.

#### *C. Controlling at Physical Machine Level*

Autoscaling decision making on each PM independently is referred to as PM level control. The primary intention of PM level control is to manage the QoS interference caused by co-hosted VMs. Among the others, Xu et al. [37] control the VMs collectively at the PM level, in this way, it tries to promote better management of QoS interference. The extended work from Xu et al. [52] consider QoS interference at the VM level, therefore the granularity of control is based on each PM. Similarly, Bu et al. [9] considers VM level QoS interference and thus the control is for each PM. Minarolli and Freisleben [85] consider all the co-hosted VM in conjunction with each others and thus its control granularity is at the PM level. Lama et al. [62] control at the PM level in order to handle QoS interference.

#### *D. Controlling at Cloud Level*

The most coarse level of control granularity is at the cloud level. The majority of the work achieves autoscaling at the cloud level by using a centralised and global controller, with

an aim to manage utility ([70], [43], [100], [31], [74], [23]), profits ([57], [35]) and availability ([50]). Among others, Ferretti et al. [17] control the QoS for all cloud-based services in a global manner. However, the actual deployment can be either centralised or decentralised. Similarly, CRAMP [44] uses a centralized and global controller, it controls the entire cloud for cost and QoS. CloudOpt [18] also controls the entire cloud using centralised control, as the considered optimisation involves all the PM in the cloud. Zhang et al. [33] control the cost of the entire cloud with respect to how the VM instances of Amazon can be utilised. BRGA [63] maintains global view of the entire cloud, and thus it belongs to cloud level control. The FOSII project [101] also controls the entire cloud in a centralised manner, where the goal is to manage the entire cloud at infrastructure level.

Some of the developments have relied on a decentralised manner where a consensus protocol is employed for controlling at the cloud granularity. For example, Wuhib et al. [46] aim to control the entire cloud, and thus the QoS and the overall power consumption of cloud can be collectively managed. In the mean time, they have relied on decentralised deployment, which can reduce the overhead of cloud-level control.

#### *E. Controlling at Multiple Levels*

Some work operates at multiple levels, with an aim to better manage the overhead and global benefit [102], [103], [104]. For example, Minarolli and Freisleben [45] combine both PM level and cloud level control, where the PM level is decentralised and the objective is to optimise the utility locally. Similarly, SmartSLA [19] aims to control the resource allocation for all the cloud-based services, therefore it utilises a global, cloud-level control in addition to the decentralised local control on each VM.

In summary, the finer granularity of control implies that it is harder to achieve globally-optimal benefit but likely to generate smaller overhead. On the other hand, globally-optimal benefit can be easier reached with large overhead if the granularity of control is coarser. All of the approaches surveyed operate at static and fixed granularity of control, even for the hybrid ones. As a result, given the time-varying QoS sensitivity and interference in cloud, they can be inflexible for any runtime changes about the effects of control granularity to the global benefit.

### **VI. TRADE-OFF DECISION MAKING**

The final important logical aspect in cloud autoscaling is the challenging decision making process, with the goal to optimise QoS and cost objectives. It is even harder to handle the trade-off between possibly conflicting objectives. Such decision making process is essentially a combinatorial optimisation problem where the output is the optimal decision containing the newly configured values for all related control primitives. In the following, we survey the key work on the decision making for cloud autoscaling. In particular, we classify them into three categories, these are Rule Based Control, Control Theoretic Approach and Search Based Optimisation. The taxonomy has been illustrated in Figure 5.

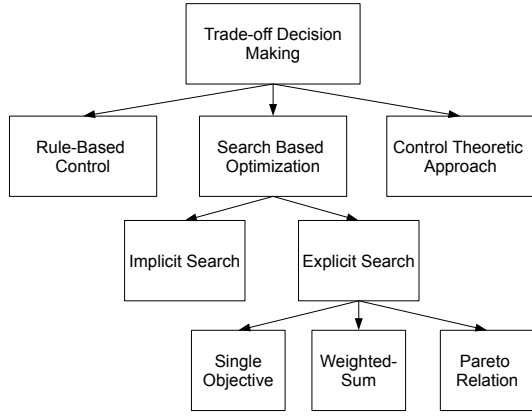


Fig. 5. The Taxonomy of Trade-off Decision Making for Autoscaling in Cloud.

#### A. Rule Based Control

Rule-based control is the most classic approaches for making decision in cloud autoscaling. Commonly, one or more conditions are manually specified and mapped to a decision, e.g., increase CPU and memory by  $x$  if the throughput is lower than  $y$ . Therefore, the possible trade-off is often implicitly handled by the conditions and actions mapping. Specifically, Cloudline [31] allows programmable elasticity rules to drive autoscaling decisions. It is also possible to modify these rules at runtime as required by the users. Copil et al. [71] handle the decision making process by specifying different condition-and-actions mapping for autoscaling in the cloud. In addition, the rules can be defined at different levels, e.g., PaaS and IaaS. Similarly, Ferretti et al. [17] allow to setup mapping between QoS expectation and actions using XML like notations. The autoscaling decision making in the work from Emeakaroha et al. [50] is also based on predefined rules, in addition, a simple heuristic algorithm is used to search for the best available VMs. Rule based control for autoscaling decision making can be also found in other work, e.g., Wuhib et al. [46], Manuer et al. [101], Han et al. [32] and Chazalet et al. [58].

Overall, we discovered that all the rule based autoscaling decision making approaches have considered both vertical and horizontal scaling as the final actuations in the cloud. Generally, rule based control is a highly intuitive approach for autoscaling decision making, and it also has negligible overhead. However, the static nature of the rules requires to assume all the possible conditions and the effects of those decisions that are mapped to the conditions. In addition, the fact that they heavily rely on human intervention and analysis can quickly become an issue. Consequently, they tend to be limited in dealing with the dynamics and uncertainties in cloud, especially when there are complex trade-offs.

#### B. Control Theoretic Approach

Advanced control theory is another widely investigated approach for autoscaling decision making in cloud because of its low latency and dynamic nature. However, it is difficult to explicitly reason about the effects of possible trade-off

decisions in a control theoretic approach.

Among the others, classical controllers (e.g., Proportional-Derivative control [43] [40] [44], Kalman control [42] [24] and Fuzzy control [38] [41] [51]) are commonly designed as a sole approach to make autoscaling decisions in the cloud. Specifically, ARUVE [43] and CRAMP [44] utilises a Proportional-Derivative (PD) controller, where the proportional and derivative factors do not depend on a QoS model of the application or the infrastructure dynamics, and support proactive resource allocation for the application server tier with dynamic scaling of web applications in a shared hosting environment. Anglano et al. [38] and Albano et al. [41] apply fuzzy control that is updated by fuzzy rules at runtime. The aim is to optimise both QoS, cost and energy by autoscaling hardware resources. Although the authors claim they can cope with any hardware resources, the approach only focus on CPU allocation. They have also assumed that QoS interference rarely occurs. Kalyvianaki et al. [42] and Grandhi et al. [24] use MIMO model and Kalman controller for making autoscaling decisions, and the authors aim at response time by autoscaling CPU on a VM.

Control theoretic approaches can be sometime used with other algorithms to better facilitate the autoscaling decision making [47], [45], [39], [105], [53], [62]. Particularly, the gains in the controllers can be further tuned by optimisation and/or machine learning algorithms, and this is especially useful for Model Predictive Control (MPC). Among others, Zhu and Agrawal [53] utilise a Proportional-Integral-Derivative (PID) and reinforcement learning controller for decision making with respect to adapting software control primitives. Such result is then tuned in conjunction with the hardware control primitives using exhaustive search. The QoS attributes and cost are formulated as weighted-sum relation. The autoscaling decision making process in APPLEware [62] have relied on MPC, which involves optimising a cost function that expresses the local control objectives and resource constraints over a time interval. The current state of the local application and the control decisions made by neighbouring controllers of a VM are taken into account to perform the optimisation using quadratic programming solver.

In summary, we discovery that the majority of the control theoretic work have considered both vertical and horizontal scaling as the final actuations in the cloud. Overall, the control theoretic approaches are efficient for making autoscaling decisions. However, the major drawback of control theoretic approaches is that they require to make many actuations on the physical system, in order to collect the 'errors' for stabilising itself. This means that amateur decisions are very likely to be made. In addition, the trade-offs are only implicitly handled.

#### C. Search Based Optimisation

A large amount of existing work relies on search-based optimisation, in which the decisions and trade-offs are extensively reasoned in a finite, but possibly large search space. Depending on the algorithms, search-based optimisation for autoscaling decision making in the cloud can be either *explicit* or *implicit*—the former performs optimisation as guided by

explicit system models; while this process is not required for the later.

### *Implicit search*

As mentioned, the implicit and search-based optimisation approaches for autoscaling decision making do not use QoS models. Similar to the control theoretic approaches, the implicit search is also limited in reasoning about the possible trade-offs. For example, the work from Xu et al. [37], [52] applies a model-free Reinforcement Learning (RL) approach for adapting thread, CPU and memory for QoS and cost. The approach is however implicit, providing that there is neither explicit system models nor explicit optimisation. The authors have considered QoS interference during autoscaling. Similarly, VScale [36] utilises RL for making autoscaling decisions, which are then achieved by vertical scaling. The RL is realised by using parallel learning, that is to say, the authors intend to speed up agent's learning process of approximated model by learning in parallel. Therefore, the agent does not have to visit every state-action pair in a given environment.

The approaches that rely on demand prediction (e.g., the Autoflex [106], PRESS [30], [97], [107], [29] and [108]) are also regarded as implicit search. This is because the autoscaling decision is essentially predicted by the demand models, without the needs of reasoning or optimisation process.

### *Explicit search*

In search-based optimisation category, the explicit approaches for autoscaling decision making rely on the explicit QoS models to evaluate and guide the search process. Depending on the different formulations of the decision making problem for autoscaling in the cloud, explicit search can reason about the effects of decisions and the possible trade-offs in details. In this paper, we have surveyed the approaches that rely on three the most commonly used formulations, these are single objective optimisation, weighted-sum optimisation, and Pareto optimisation.

It is common to optimise only a single objective (e.g., cost or profit) for autoscaling in the cloud, providing that the requirements of the other objectives are satisfied (i.e., they are often regarded as constraints) [98], [33], [48], [66], [80], [54], [27], [97], [18], [109], [49], [11]. For example, Kingfisher [27] and Sedaghat et al. [97] use Integer Linear Programming (ILP) to optimise the cost for scaling the CPU and memory for VMs of an application while regarding the demand for satisfying QoS as constraint. Sedaghat et al. [97] has additionally assumed fixed VM bundles. Similarly, CloudOpt [18] models the decision making for autoscaling using a weighted-sum of different aspect of cost, which is still regarded as single objective optimisation, and the optimisation is then resolved by mixed integer programming approach. A recent extension of ARUVE [109] formulates the decision making in autoscaling as a single optimisation on cost, which is resolved by using Ant Colony Optimisation (ACO).

To apply search based optimisation for autoscaling in the cloud, the most widely solution for handling the multi-objectivity is to aggregate all related objectives into a weighted

(usually weighted-sum) formulation, which converts the decision making process into a single objective optimisation problem. The search based algorithm include: exhaustive search [70] [61] [28] [25], auxiliary network flow model [57], force-directed search [67], binary search [34]. For example, the FoSII project [59] [60] regards the autoscaling decision making as case based reasoning process, where the decision is made by looking for similar cases from the past and reusing the solutions of these cases to solve the current one. The case and solution pairs are linked to aggregative utility values based on the analytical model, thus the reasoning process is essentially an optimisation for the optimal decision using exhaustive search algorithm. Goudarzi and Pedram [67] use a weighted-sum formulation containing QoS and cost for an application tier. The optimisation is resolved using force-directed search, in which an initial solution based on the solution given for the profit upper bound problem is generated. Next, distribution rates are fixed and resource sharing is improved by a local optimisation step.

Some work has relied on more advanced and nonlinear search algorithms, ranging from relatively simple ones: dynamic programming [23] and local-search strategy [74], to more complex forms: grid search [19], decision tree search [26] [11] and quadratic programming [20]. For example, CLOUDFARM [23] addresses the decision making based on a weighted-sum utility function of all cloud-based application and services. The decision making process is formulated as a knapsack problem, which can be resolved by dynamic programming. In SmartSLA [19], the decision making for autoscaling is aimed for optimising SLA penalty, which is essentially based on the aggregation of expected QoS values. The optimisation is resolved using a grid search algorithm. To optimise weighted-sum utilisation, Amiya et al. [11] have explicitly aimed to mitigate QoS interference in the cloud using heuristic based decision tree search, however, they only intend to autoscale software control primitives.

Metaheuristic algorithms are also popular for autoscaling decision making in the cloud, because they can often efficiently address NP-hard problems with approximated results. The most common algorithms include: Tabu Search [35], Genetic Algorithm (GA) [10] [85] [63], Particle Swarm Optimisation (PSO) [10]. As an example, Zhu et al. [35] formulate the autoscaling decision making as optimise a weighted-sum formulation of response time and cost. To optimise the objectives, the authors apply a hybrid Tabu Search, which, in every iteration, the current matrix is disturbed and a new decision is generated as initial solution of gradient descent. After reaching a particular fixed point, the variation of profit is calculated and the best configuration is returned.

Finally, Pareto relation can explicitly handle multi-objectivity for autoscaling in the cloud without the need to specify weights on the objectives [21], [69], [65], [99], [110]. For example, Kateb et al. [21] formulate the decision making process as Pareto-based multi-objective optimisation, which is solved by Multi-objective Genetic Algorithm (MOGA, e.g., NSGA-II). At each generation, MOGA identifies non-dominating solutions. Crowding distance is used to calculate

the distance between an individual and its neighbours. In particular, each generation of the search is evaluated using epsilon dominance which is a relaxed form of the commonly used Pareto-dominance metric. In E<sup>3</sup>-R [65], the decision making problem is formulated as Pareto front, where it is resolved by using MOGA. In addition, the approach applies objective reduction technique with an aim to remove the objectives, which are not significantly conflicted with the others, from the decision making process. In this way, the author aims to reduce the overhead while not affecting the quality of decisions.

In conclusion, we discover that around two thirds of the surveyed search based approaches have considered both vertical and horizontal scaling as the final actions in the cloud. Overall, the nature of search-based optimisation permits certain level of reasoning during the decision making, and this presumably provides better assurance on the quality of the decisions before conducting the actual scaling actions. As we can see, there are small amount of the approaches surveyed belong to implicit search, which can be efficient as there is no need for QoS modelling. Nevertheless, the absence of QoS models also mean that they cannot explicitly handle the trade-offs. On the other hand, the other approaches make use of the explicit search, however, majority of those work formulate the problem as single objective or rely on weighted-sum of objectives, and hence their search of possible trade-offs decision tend to be limited in terms of both optimality and diversity. A limited amount of effort has considered Pareto relation, however, none of them have considered well-compromised trade-off, i.e., the decisions that have balanced improvements on the related objectives. Finally, QoS interference is often ignored in autoscaling decision making, even if it is considered, there is no explicit solution for handling the related trade-offs.

## VII. CONCLUSION

In this paper, we described the background and definitions for cloud autoscaling, self-aware and self-adaptive systems in general. Subsequently, we outlined the major requirements for the key logical aspects of autoscaling in the cloud, and discuss the key state-of-the-art developments proposed for each of the logical aspects, from which a taxonomy is provided. We hope that our systematic survey and taxonomy will motivate further research for more intelligent cloud autoscaling and its interaction with the other problems in the cloud.

## REFERENCES

- [1] T. Chen and R. Bahsoon, "Towards a smarter cloud: Self-aware autoscaling of cloud configurations and resources," *Computer, IEEE*, vol. 48, no. 9, Sept 2015.
- [2] B. Cheng, R. de Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic, G. Di Marzo Serugendo, S. Dustdar, A. Finkelstein, C. Gacek, K. Geihs, V. Grassi, G. Karsai, H. Kienle, J. Kramer, M. Litoiu, S. Malek, R. Mirandola, H. Miller, S. Park, M. Shaw, M. Tichy, M. Tivoli, D. Weyns, and J. Whittle, "Software engineering for self-adaptive systems: A research roadmap," in *Software Engineering for Self-Adaptive Systems*, ser. Lecture Notes in Computer Science, B. Cheng, R. de Lemos, H. Giese, P. Inverardi, and J. Magee, Eds. Springer Berlin Heidelberg, 2009, vol. 5525, pp. 1–26.
- [3] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and research challenges," *ACM Trans. Auton. Adapt. Syst.*, vol. 4, no. 2, pp. 14:1–14:42, May 2009. [Online]. Available: <http://doi.acm.org/10.1145/1516533.1516538>
- [4] T. Chen, F. Faniyi, R. Bahsoon, P. R. Lewis, X. Yao, L. L. Minku, and L. Esterle, "The handbook of engineering self-aware and self-expressive systems," *arXiv preprint arXiv:1409.1793*, 2014.
- [5] P. R. Lewis, A. Chandra, F. Faniyi, K. Glette, T. Chen, R. Bahsoon, J. Torresen, and X. Yao, "Architectural aspects of self-aware and self-expressive computing systems: From psychology to engineering," *Computer*, vol. 48, no. 8, pp. 62–70, Aug 2015.
- [6] P. Lewis, A. Chandra, S. Parsons, E. Robinson, K. Glette, R. Bahsoon, J. Torresen, and X. Yao, "A survey of self-awareness and its application in computing systems," in *Self-Adaptive and Self-Organizing Systems Workshops (SASOW)*, 2011 Fifth IEEE Conference on, Oct 2011, pp. 102–107.
- [7] T. Chen, F. Faniyi, and R. Bahsoon, *Design Patterns and Primitives: Introduction of Components and Patterns for SACS*. Cham: Springer International Publishing, 2016, pp. 53–78.
- [8] T. Becker, A. Agne, P. Lewis, R. Bahsoon, F. Faniyi, L. Esterle, A. Keller, A. Chandra, A. Jensenius, and S. Stikkerich, "Epics: Engineering proprioception in computing systems," in *Computational Science and Engineering (CSE)*, 2012 IEEE 15th International Conference on, Dec 2012, pp. 353–360.
- [9] X. Bu, J. Rao, and C. zhong Xu, "Coordinated self-configuration of virtual machines and appliances using a model-free learning approach," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 24, no. 4, pp. 681–690, April 2013.
- [10] Y. Zhang, G. Huang, X. Liu, and H. Mei, "Integrating resource consumption and allocation for infrastructure resources on-demand," in *Cloud Computing (CLOUD)*, 2010 IEEE 3rd International Conference on, July 2010, pp. 75–82.
- [11] A. K. Maji, S. Mitra, B. Zhou, S. Bagchi, and A. Verma, "Mitigating interference in cloud services by middleware reconfiguration," in *Proceedings of the 15th International Middleware Conference*, ser. Middleware '14. New York, NY, USA: ACM, 2014, pp. 277–288. [Online]. Available: <http://doi.acm.org/10.1145/2663165.2663330>
- [12] H. Ghanbari, B. Simmons, M. Litoiu, and G. Iszlai, "Exploring alternative approaches to implement an elasticity policy," in *Cloud Computing (CLOUD)*, 2011 IEEE International Conference on, July 2011, pp. 716–723.
- [13] N. Rupal, K. Aman, and G. Alireza, "Q-clouds: Managing performance interference effects for qos-aware clouds," in *Proceedings of the 5th European Conference on Computer Systems*, ser. EuroSys '10. New York, NY, USA: ACM, 2010, pp. 237–250. [Online]. Available: <http://doi.acm.org/10.1145/1755913.1755938>
- [14] Y. Brun, G. Marzo Serugendo, C. Gacek, H. Giese, H. Kienle, M. Litoiu, H. Müller, M. Pezzè, and M. Shaw, "Software engineering for self-adaptive systems," B. H. Cheng, R. Lemos, H. Giese, P. Inverardi, and J. Magee, Eds. Berlin, Heidelberg: Springer-Verlag, 2009, ch. Engineering Self-Adaptive Systems Through Feedback Loops, pp. 48–70.
- [15] H. Hoffman, "Seec: A framework for self-aware management of goals and constraints in computing systems (power-aware computing, accuracy-aware computing, adaptive computing, autonomic computing)," Ph.D. dissertation, Cambridge, MA, USA, 2013, aAI0829261.
- [16] IBM, "An architectural blueprint for autonomic computing," *IBM Technical Report*, 2003.
- [17] S. Ferretti, V. Ghini, F. Panzieri, M. Pellegrini, and E. Turrini, "Qos-aware clouds," in *Cloud Computing (CLOUD)*, 2010 IEEE 3rd International Conference on, July 2010, pp. 321–328.
- [18] J. Li, M. Woodside, J. Chinneck, and M. Litoiu, "Cloudbot: Multi-goal optimization of application deployments across a cloud," in *Network and Service Management (CNSM)*, 2011 7th International Conference on, Oct 2011, pp. 1–9.
- [19] P. Xiong, Y. Chi, S. Zhu, H. J. Moon, C. Pu, and H. Hacgumus, "Smartsla: Cost-sensitive management of virtualized resources for cpu-bound database services," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 26, no. 5, pp. 1441–1451, May 2015.
- [20] P. Padala, K.-Y. Hou, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant, "Automated control of multiple virtualized resources," in *Proceedings of the 4th ACM European Conference on Computer Systems*, ser. EuroSys '09. New York, NY, USA: ACM, 2009, pp. 13–26. [Online]. Available: <http://doi.acm.org/10.1145/1519065.1519068>

- [21] D. El Kateb, F. Fouquet, G. Nain, J. A. Meira, M. Ackerman, and Y. Le Traon, "Generic cloud platform multi-objective optimization leveraging models@run.time," in *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, ser. SAC '14. New York, NY, USA: ACM, 2014, pp. 343–350. [Online]. Available: <http://doi.acm.org/10.1145/2554850.2555044>
- [22] J. Jiang, J. Lu, and G. Zhang, "An innovative self-adaptive configuration optimization system in cloud computing," in *Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference on*, Dec 2011, pp. 621–627.
- [23] V. Nikolov, S. Kachele, F. Hauck, and D. Rautenbach, "Cloudfarm: An elastic cloud platform with flexible and adaptive resource management," in *Utility and Cloud Computing (UCC), 2014 IEEE/ACM 7th International Conference on*, Dec 2014, pp. 547–553.
- [24] A. Gandhi, P. Dube, A. Karve, A. Kochut, and L. Zhang, "Adaptive, model-driven autoscaling for cloud applications," in *11th International Conference on Autonomic Computing*, 2014.
- [25] J. Jiang, J. Lu, G. Zhang, and G. Long, "Optimal cloud resource auto-scaling for web applications," in *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, May 2013, pp. 58–65.
- [26] H. Fernandez, G. Pierre, and T. Kielmann, "Autoscaling web applications in heterogeneous cloud infrastructures," in *Cloud Engineering (IC2E), 2014 IEEE International Conference on*, March 2014, pp. 195–204.
- [27] U. Sharma, P. Shenoy, S. Sahu, and A. Shaikh, "A cost-aware elasticity provisioning system for the cloud," in *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*, June 2011, pp. 559–570.
- [28] A. Gambi, G. Toffetti, C. Pautasso, and M. Pezze, "Kriging controllers for cloud applications," *Internet Computing, IEEE*, vol. 17, no. 4, pp. 40–47, July 2013.
- [29] H. Chihi, W. Chainbi, and K. Ghedira, "An energy-efficient self-provisioning approach for cloud resources management," *SIGOPS Oper. Syst. Rev.*, vol. 47, no. 3, pp. 2–9, Nov. 2013. [Online]. Available: <http://doi.acm.org/10.1145/2553070.2553072>
- [30] Z. Gong, X. Gu, and J. Wilkes, "Press: Predictive elastic resource scaling for cloud systems," in *Network and Service Management (CNSM), 2010 International Conference on*, Oct 2010, pp. 9–16.
- [31] G. Galante and L. Bona, "Constructing elastic scientific applications using elasticity primitives," in *Computational Science and Its Applications ICCSA 2013*, ser. Lecture Notes in Computer Science, B. Murgante, S. Misra, M. Carlini, C. Torre, H.-Q. Nguyen, D. Taniar, B. Apduhan, and O. Gervasi, Eds. Springer Berlin Heidelberg, 2013, vol. 7975, pp. 281–294.
- [32] R. Han, L. Guo, M. Ghanem, and Y. Guo, "Lightweight resource scaling for cloud applications," in *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*, May 2012, pp. 644–651.
- [33] Q. Zhang, Q. Zhu, and R. Boutaba, "Dynamic resource allocation for spot markets in cloud computing environments," in *Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on*, Dec 2011, pp. 178–185.
- [34] F. Kabir and D. Chiu, "Reconciling cost and performance objectives for elastic web caches," in *Cloud and Service Computing (CSC), 2012 International Conference on*, Nov 2012, pp. 88–95.
- [35] Z. Zhu, J. Bi, H. Yuan, and Y. Chen, "Sla based dynamic virtualized resources provisioning for shared cloud data centers," in *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, July 2011, pp. 630–637.
- [36] L. Yazdanov and C. Fetzer, "Vscaler: Autonomic virtual machine scaling," in *Proceedings of the 2013 IEEE Sixth International Conference on Cloud Computing*, ser. CLOUD '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 212–219. [Online]. Available: <http://dx.doi.org/10.1109/CLOUD.2013.142>
- [37] J. Rao, X. Bu, C.-Z. Xu, and K. Wang, "A distributed self-learning approach for elastic provisioning of virtualized cloud resources," in *Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS), 2011 IEEE 19th International Symposium on*, July 2011, pp. 45–54.
- [38] C. Anglano, M. Canonico, and M. Guazzone, "Fc2q: exploiting fuzzy control in server consolidation for cloud applications with sla constraints," *Concurrency and Computation: Practice and Experience*, pp. n/a–n/a, 2014. [Online]. Available: <http://dx.doi.org/10.1002/cpe.3410>
- [39] Y. Guo, P. Lama, C. Jiang, and X. Zhou, "Automated and agile server parametertuning by coordinated learning and control," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 25, no. 4, pp. 876–886, April 2014.
- [40] H. C. Lim, S. Babu, J. S. Chase, and S. S. Parekh, "Automated control in cloud computing: Challenges and opportunities," in *Proceedings of the 1st Workshop on Automated Control for Datacenters and Clouds*, ser. ACDC '09. New York, NY, USA: ACM, 2009, pp. 13–18. [Online]. Available: <http://doi.acm.org/10.1145/1555271.1555275>
- [41] L. Albano, C. Anglano, M. Canonico, and M. Guazzone, "Fuzzy-q amp: Achieving qos guarantees and energy savings for cloud applications with fuzzy control," in *Cloud and Green Computing (CGC), 2013 Third International Conference on*, Sept 2013, pp. 159–166.
- [42] E. Kalyvianaki, T. Charalambous, and S. Hand, "Adaptive resource provisioning for virtualized servers using kalman filters," *ACM Trans. Auton. Adapt. Syst.*, vol. 9, no. 2, pp. 10:1–10:35, Jul. 2014. [Online]. Available: <http://doi.acm.org/10.1145/2626290>
- [43] A. Ashraf, B. Byholm, J. Lehtinen, and I. Porres, "Feedback control algorithms to deploy and scale multiple web applications per virtual machine," in *Software Engineering and Advanced Applications (SEAA), 2012 38th EUROMICRO Conference on*, Sept 2012, pp. 431–438.
- [44] A. Ashraf, B. Byholm, and I. Porres, "Cramp: Cost-efficient resource allocation for multiple web applications with proactive scaling," in *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*, Dec 2012, pp. 581–586.
- [45] D. Minarolli and B. Freisleben, "Virtual machine resource allocation in cloud computing via multi-agent fuzzy control," in *Cloud and Green Computing (CGC), 2013 Third International Conference on*, Sept 2013, pp. 188–194.
- [46] F. Wuhib, R. Stadler, and H. Lindgren, "Dynamic resource allocation with management objectives: Implementation for an openstack cloud," in *Network and service management (cnsm), 2012 8th international conference and 2012 workshop on systems virtualization management (svm)*, Oct 2012, pp. 309–315.
- [47] P. Xiong, Z. Wang, S. Malkowski, Q. Wang, D. Jayasinghe, and C. Pu, "Economic and robust provisioning of n-tier cloud workloads: A multi-level control approach," in *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*, June 2011, pp. 571–580.
- [48] S. Kim, J.-S. Kim, S. Hwang, and Y. Kim, "An allocation and provisioning model of science cloud for high throughput computing applications," in *Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference*, ser. CAC '13. New York, NY, USA: ACM, 2013, pp. 27:1–27:8. [Online]. Available: <http://doi.acm.org/10.1145/2494621.2494649>
- [49] R. C. Chiang, J. Hwang, H. H. Huang, and T. Wood, "Matrix: Achieving predictable virtual machine performance in the clouds," in *11th International Conference on Autonomic Computing*, 2014.
- [50] V. Emeakaroha, I. Brandic, M. Maurer, and I. Breskovic, "Sla-aware application deployment and resource allocation in clouds," in *Computer Software and Applications Conference Workshops (COMPSACW), 2011 IEEE 35th Annual*, July 2011, pp. 298–303.
- [51] L. Wang, J. Xu, and M. Zhao, "Application-aware cross-layer virtual machine resource management," in *Proceedings of the 9th International Conference on Autonomic Computing*, ser. ICAC '12. New York, NY, USA: ACM, 2012, pp. 13–22. [Online]. Available: <http://doi.acm.org/10.1145/2371536.2371541>
- [52] C.-Z. Xu, J. Rao, and X. Bu, "Url: A unified reinforcement learning approach for autonomic cloud management," *Journal of Parallel and Distributed Computing*, vol. 72, no. 2, pp. 95 – 105, 2012. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0743731511001924>
- [53] Q. Zhu and G. Agrawal, "Resource provisioning with budget constraints for adaptive applications in cloud environments," *Services Computing, IEEE Transactions on*, vol. 5, no. 4, pp. 497–511, Fourth 2012.
- [54] N. Huber, F. Brosig, and S. Kounev, "Model-based self-adaptive resource allocation in virtualized environments," in *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS '11. New York, NY, USA: ACM, 2011, pp. 90–99. [Online]. Available: <http://doi.acm.org/10.1145/1988008.1988021>
- [55] F. Sironi, M. Triverio, H. Hoffmann, M. Maggio, and M. Santambrogio, "Self-aware adaptation in fpga-based systems," in *Field Programmable Logic and Applications (FPL), 2010 International Conference on*, Aug 2010, pp. 187–192.

- [56] C. Bolchini, M. Carminati, A. Miele, and E. Quintarelli, "A framework to model self-adaptive computing systems," in *Adaptive Hardware and Systems (AHS), 2013 NASA/ESA Conference on*, June 2013, pp. 71–78.
- [57] J. Li, J. Chinneck, M. Woodside, M. Litoiu, and G. Iszlai, "Performance model driven qos guarantees and optimization in clouds," in *Software Engineering Challenges of Cloud Computing, 2009. CLOUD '09. ICSE Workshop on*, May 2009, pp. 15–22.
- [58] A. Chazalet, F. D. Tran, M. Deslaugiers, A. Lefebvre, F. Exertier, and J. Legrand, "Adding self-scaling capability to the cloud to meet service level agreements," *International Journal on Advances in Intelligent Systems*, vol. 4, no. 3, 2011.
- [59] I. Brandic, V. Emeakaroha, M. Maurer, S. Dustdar, S. Acs, A. Kertesz, and G. Kecskemeti, "Laysi: A layered approach for sla-violation propagation in self-manageable cloud infrastructures," in *Computer Software and Applications Conference Workshops (COMPSACW), 2010 IEEE 34th Annual*, July 2010, pp. 365–370.
- [60] M. Maurer, I. Brandic, V. Emeakaroha, and S. Dustdar, "Towards knowledge management in self-adaptable clouds," in *Services (SERVICES-1), 2010 6th World Congress on*, July 2010, pp. 527–534.
- [61] R. Calinescu, L. Grunske, M. Kwiatkowska, R. Mirandola, and G. Tamburrelli, "Dynamic qos management and optimization in service-based systems," *Software Engineering, IEEE Transactions on*, vol. 37, no. 3, pp. 387–409, May 2011.
- [62] P. Lama, Y. Guo, and X. Zhou, "Autonomic performance and power control for co-located web applications on virtualized servers," in *Quality of Service (IWQoS), 2013 IEEE/ACM 21st International Symposium on*, June 2013, pp. 1–10.
- [63] O. Abdul-Rahman, M. Munetomo, and K. Akama, "Toward a genetic algorithm based flexible approach for the management of virtualized application environments in cloud platforms," in *Computer Communications and Networks (ICCCN), 2012 21st International Conference on*, July 2012, pp. 1–9.
- [64] T. Llorido-Botran, J. Miguel-Alonso, and J. Lozano, "A review of auto-scaling techniques for elastic applications in cloud environments," *Journal of Grid Computing*, vol. 12, no. 4, pp. 559–592, 2014.
- [65] H. Wada, J. Suzuki, Y. Yamano, and K. Oba, "Evolutionary deployment optimization for service-oriented clouds," *Softw. Pract. Exper.*, vol. 41, no. 5, pp. 469–493, Apr. 2011. [Online]. Available: <http://dx.doi.org/10.1002/spe.1032>
- [66] C. Wang, J. Chen, B. B. Zhou, and A. Zomaya, "Just satisfactory resource provisioning for parallel applications in the cloud," in *Services (SERVICES), 2012 IEEE Eighth World Congress on*, June 2012, pp. 285–292.
- [67] H. Goudarzi and M. Pedram, "Multi-dimensional sla-based resource allocation for multi-tier cloud computing systems," in *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, July 2011, pp. 324–331.
- [68] J. Bi, Z. Zhu, R. Tian, and Q. Wang, "Dynamic provisioning modeling for virtualized multi-tier applications in cloud data center," in *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, July 2010, pp. 370–377.
- [69] H. Li, G. Casale, and T. Ellahi, "Sla-driven planning and optimization of enterprise applications," in *Proceedings of the First Joint WOSP/SIPEW International Conference on Performance Engineering*, ser. WOSP/SIPEW '10. New York, NY, USA: ACM, 2010, pp. 117–128. [Online]. Available: <http://doi.acm.org/10.1145/1712605.1712625>
- [70] R. Chi, Z. Qian, and S. Lu, "A game theoretical method for auto-scaling of multi-tiers web applications in cloud," in *Proceedings of the Fourth Asia-Pacific Symposium on Internetwork*, ser. Internetwork '12. New York, NY, USA: ACM, 2012, pp. 3:1–3:10. [Online]. Available: <http://doi.acm.org/10.1145/2430475.2430478>
- [71] G. Copil, D. Moldovan, H.-L. Truong, and S. Dustdar, "Multi-level elasticity control of cloud services," in *Service-Oriented Computing*, ser. Lecture Notes in Computer Science, S. Basu, C. Pautasso, L. Zhang, and X. Fu, Eds. Springer Berlin Heidelberg, 2013, vol. 8274, pp. 429–436.
- [72] V. C. Emeakaroha, R. N. Calheiros, M. A. S. Netto, I. Brandic, and C. A. F. D. Rose, "Desvi: An architecture for detecting sla violations in cloud computing infrastructures," in *Proceedings of the 2nd international ICST conference on Cloud computing (CloudComp'10)*, 2010.
- [73] V. Emeakaroha, I. Brandic, M. Maurer, and S. Dustdar, "Low level metrics to high level slas - lom2his framework: Bridging the gap between monitored metrics and sla parameters in cloud environments," in *High Performance Computing and Simulation (HPCS), 2010 International Conference on*, June 2010, pp. 48–54.
- [74] B. Addis, D. Ardagna, B. Panicucci, and L. Zhang, "Autonomic management of cloud service centers with availability guarantees," in *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, July 2010, pp. 220–227.
- [75] F. Fittkau, S. Frey, and W. Hasselbring, "Cdosim: Simulating cloud deployment options for software migration support," in *Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA), 2012 IEEE 6th International Workshop on the*, Sept 2012, pp. 37–46.
- [76] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "Cloudsim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Softw. Pract. Exper.*, vol. 41, no. 1, pp. 23–50, Jan. 2011. [Online]. Available: <http://dx.doi.org/10.1002/spe.995>
- [77] B. Wickremasinghe, R. Calheiros, and R. Buyya, "Cloudanalyst: A cloudsimsim-based visual modeller for analysing cloud computing environments and applications," in *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*, April 2010, pp. 446–452.
- [78] G. Keller, M. Tighe, H. Lutfiyya, and M. Bauer, "Desim: A data centre simulation tool," in *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*, May 2013, pp. 1090–1091.
- [79] Y. Diao, N. Gandhi, J. Hellerstein, S. Parekh, and D. Tilbury, "Using mimo feedback control to enforce policies for interrelated metrics with application to the apache web server," in *Network Operations and Management Symposium, 2002. NOMS 2002. 2002 IEEE/IFIP*, 2002, pp. 219–234.
- [80] X. J. Collazo-Mojica, S. Sadjadi, J. Ejarque, and R. M. Badia, "Cloud application resource mapping and scaling based on monitoring of qos constraints." San Francisco, United States: Knowledge Systems Institute Graduate School, Jul 2012, Conference Paper, p. 88–93.
- [81] T. Zheng, M. Litoiu, and M. Woodside, "Integrated estimation and tracking of performance model parameters with autoregressive trends," in *Proceedings of the 2Nd ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '11. New York, NY, USA: ACM, 2011, pp. 157–166. [Online]. Available: <http://doi.acm.org/10.1145/1958746.1958772>
- [82] H. Ghanbari, C. Barna, M. Litoiu, M. Woodside, T. Zheng, J. Wong, and G. Iszlai, "Tracking adaptive performance models using dynamic clustering of user classes," *SIGSOFT Softw. Eng. Notes*, vol. 36, no. 5, pp. 179–188, Sep. 2011. [Online]. Available: <http://doi.acm.org/10.1145/1958746.1958774>
- [83] L. Wang, J. Xu, and M. Zhao, "Tracking adaptive performance models using dynamic clustering of user classes," in *7th International Workshop on Feedback Computing*, 2012.
- [84] S. Kundu, R. Rangaswami, A. Gulati, M. Zhao, and K. Dutta, "Modeling virtualized applications using machine learning techniques," in *Proceedings of the 8th ACM SIGPLAN/SIGOPS Conference on Virtual Execution Environments*, ser. VEE '12. New York, NY, USA: ACM, 2012, pp. 3–14. [Online]. Available: <http://doi.acm.org/10.1145/2151024.2151028>
- [85] D. Minarolli and B. Freisleben, "Distributed resource allocation to virtual machines via artificial neural networks," in *Parallel, Distributed and Network-Based Processing (PDP), 2014 22nd Euromicro International Conference on*, Feb 2014, pp. 490–499.
- [86] G. Kousiouris, T. Cucinotta, and T. Varvarigou, "The effects of scheduling, workload type and consolidation scenarios on virtual machine performance and their prediction through optimized artificial neural networks," *J. Syst. Softw.*, vol. 84, no. 8, pp. 1270–1291, Aug. 2011. [Online]. Available: <http://dx.doi.org/10.1016/j.jss.2011.04.013>
- [87] P. Bodík, R. Griffith, C. Sutton, A. Fox, M. Jordan, and D. Patterson, "Statistical machine learning makes automatic control practical for internet datacenters," in *Proceedings of the 2009 Conference on Hot Topics in Cloud Computing*, ser. HotCloud'09. Berkeley, CA, USA: USENIX Association, 2009. [Online]. Available: <http://dl.acm.org/citation.cfm?id=1855533.1855545>
- [88] T. Chen and R. Bahsoon, "Self-adaptive and sensitivity-aware qos modeling for the cloud," in *Proceedings of the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 43–52. [Online]. Available: <http://dl.acm.org/citation.cfm?id=2663546.2663556>
- [89] T. Chen, R. Bahsoon, and X. Yao, "Online qos modeling in the cloud: A hybrid and adaptive multi-learners approach," in *Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*, ser. UCC '14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 327–336. [Online]. Available: <http://dx.doi.org/10.1109/UCC.2014.42>

- [90] T. Chen and R. Bahsoon, "Self-adaptive and online qos modeling for cloud-based software services," *IEEE Transactions on Software Engineering*, 2016.
- [91] G. Kousiouris, A. Menychtas, D. Kyriazis, S. Gogouvitis, and T. Varvarigou, "Dynamic, behavioral-based estimation of resource provisioning based on high-level application terms in cloud platforms," *Future Generation Computer Systems*, vol. 32, no. 0, pp. 27 – 40, 2014, special Section: The Management of Cloud Systems, Special Section: Cyber-Physical Society and Special Section: Special Issue on Exploiting Semantic Technologies with Particularization on Linked Data over Grid and Cloud Architectures. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0167739X12001057>
- [92] Y. Koh, R. Knauerhase, P. Brett, M. Bowman, Z. Wen, and C. Pu, "An analysis of performance interference effects in virtual environments," in *Performance Analysis of Systems Software, 2007. ISPASS 2007. IEEE International Symposium on*, April 2007, pp. 200–209.
- [93] P. Xiong, C. Pu, X. Zhu, and R. Griffith, "vperfguard: An automated model-driven framework for application performance diagnosis in consolidated cloud environments," in *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '13. New York, NY, USA: ACM, 2013, pp. 271–282. [Online]. Available: <http://doi.acm.org/10.1145/2479871.2479909>
- [94] R. Mian, P. Martin, F. Zulkernine, and J. L. Vazquez-Poletti, "Towards building performance models for data-intensive workloads in public clouds," in *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '13. New York, NY, USA: ACM, 2013, pp. 259–270. [Online]. Available: <http://doi.acm.org/10.1145/2479871.2479908>
- [95] W. Lloyd, S. Pallickara, O. David, J. Lyon, M. Arabi, and K. Rojas, "Performance modeling to support multi-tier application deployment to infrastructure-as-a-service clouds," in *Utility and Cloud Computing (UCC), 2012 IEEE Fifth International Conference on*, Nov 2012, pp. 73–80.
- [96] R. Chiang and H. Huang, "Tracon: Interference-aware scheduling for data-intensive applications in virtualized environments," in *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for*, Nov 2011, pp. 1–12.
- [97] M. Sedaghat, F. Hernandez-Rodriguez, and E. Elmroth, "A virtual machine re-packing approach to the horizontal vs. vertical elasticity trade-off for cloud autoscaling," in *Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference*, ser. CAC '13. New York, NY, USA: ACM, 2013, pp. 6:1–6:10. [Online]. Available: <http://doi.acm.org/10.1145/2494621.2494628>
- [98] J. Yang, C. Liu, Y. Shang, B. Cheng, Z. Mao, C. Liu, L. Niu, and J. Chen, "A cost-aware auto-scaling approach using the workload prediction in service clouds," *Information Systems Frontiers*, vol. 16, no. 1, pp. 7–18, 2014.
- [99] S. Frey, F. Fittkau, and W. Hasselbring, "Search-based genetic optimization for deployment and reconfiguration of software in the cloud," in *Proceedings of the 2013 International Conference on Software Engineering*, ser. ICSE '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 512–521. [Online]. Available: <http://dl.acm.org/citation.cfm?id=2486788.2486856>
- [100] T. Chen and R. Bahsoon, "Scalable service oriented replication in the cloud," in *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, July 2011, pp. 766–767.
- [101] M. Maurer, I. Brandic, and R. Sakellariou, "Self-adaptive and resource-efficient sla enactment for cloud computing infrastructures," in *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, June 2012, pp. 368–375.
- [102] T. Chen, R. Bahsoon, and G. Theodoropoulos, "Dynamic qos optimization architecture for cloud-based dddas," *Procedia Computer Science*, vol. 18, pp. 1881 – 1890, 2013, 2013 International Conference on Computational Science. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S1877050913005000>
- [103] T. Chen and R. Bahsoon, "Symbiotic and sensitivity-aware architecture for globally-optimal benefit in self-adaptive cloud," in *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS 2014. New York, NY, USA: ACM, 2014, pp. 85–94. [Online]. Available: <http://doi.acm.org/10.1145/2593929.2593931>
- [104] T. Chen, R. Bahsoon, and A.-R. H. Tawil, "Scalable service-oriented replication with flexible consistency guarantee in the cloud," *Information Sciences*, vol. 264, pp. 349 – 370, 2014. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0020025513008323>
- [105] J. Rao, Y. Wei, J. Gong, and C.-Z. Xu, "Dynaqos: Model-free self-tuning fuzzy control of virtualized resources for qos provisioning," in *Quality of Service (IWQoS), 2011 IEEE 19th International Workshop on*, June 2011, pp. 1–9.
- [106] F. Almeida Morais, F. Vilar Brasileiro, R. Vigolvino Lopes, R. Araujo Santos, W. Satterfield, and L. Rosa, "Autoflex: Service agnostic auto-scaling framework for iaas deployment models," in *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, May 2013, pp. 42–49.
- [107] E. Caron, F. Desprez, and A. Muresan, "Forecasting for grid and cloud computing on-demand resources based on pattern matching," in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, Nov 2010, pp. 456–463.
- [108] Y. Jiang, C.-S. Perng, T. Li, and R. Chang, "Cloud analytics for capacity planning and instant vm provisioning," *Network and Service Management, IEEE Transactions on*, vol. 10, no. 3, pp. 312–325, September 2013.
- [109] A. Ashraf and I. Porres, "Using ant colony system to consolidate multiple web applications in a cloud environment," in *Parallel, Distributed and Network-Based Processing (PDP), 2014 22nd Euromicro International Conference on*, Feb 2014, pp. 482–489.
- [110] T. Chen and R. Bahsoon, "Self-adaptive trade-off decision making for autoscaling cloud-based services," *IEEE Transactions on Services Computing*, 2015, doi:10.1109/TSC.2015.2499770.